Hierarchical Reasoning about Inequalities

Elisha Sacks[1]
MIT Laboratory for Computer Science
545 Technology Square, Room 370
Cambridge, MA 02139, USA

Abstract

This paper describes a program called BOUNDER that proves inequalities between functions over finite sets of constraints. Previous inequality algorithms perform well on some subset of the elementary functions, but poorly elsewhere. To overcome this problem, BOUNDER maintains a hierarchy of increasingly complex algorithms. When one fails to resolve an inequality, it tries the next. This strategy resolves more inequalities than any single algorithm. It also performs well on hard problems without wasting time on easy ones. The current hierarchy consists of four algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. Propagation is an extension of interval arithmetic that takes linear time, but ignores constraints between variables and multiple occurrences of variables. The remaining algorithms consider these factors, but require exponential time. Substitution is a new, provably correct, algorithm for utilizing constraints between variables. The final two algorithms analyze constraints between variables. Inspection examines the signs of partial derivatives. Iteration is based on several earlier algorithms from interval arithmetic.

I. Introduction

This paper describes a program called BOUNDER that proves inequalities between functions over all points satisfying a finite set of *constraints:* equalities and inequalities between functions. BOUNDER manipulates *extended elementary functions:* polynomials and compositions of exponentials, logarithms, trigonometric functions, inverse trigonometric functions, absolute values, maxima, and minima. It tests whether a set of constraints, C, implies an inequality $a \leq b$ between the extended elementary functions a and b by calculating upper and lower bounds for $a - b$ over all points satisfying C. It proves the inequality when the upper bound is negative or zero, refutes it when the lower bound is positive, and fails otherwise.

Previous bounding algorithms perform well on some subset of the extended elementary functions, but poorly elsewhere. For this reason, BOUNDER maintains a hierarchy of increasingly complex bounding algorithms. When one fails to resolve an inequality, it tries the next. Although complex algorithms derive tighter bounds than simple ones for most functions, exceptions exist. Hence, BOUNDER's hierarchy of algorithms derives tighter bounds than even its most powerful component. It also performs well on hard problems without wasting time on easier ones.

The purpose of BOUNDER is to resolve inequalities that arise in realistic modeling problems efficiently, not to derive deep theoretical results. It is an engineering utility, rather than a theorem-prover for pure mathematics. For this reason, it only addresses universally quantified inequalities, which make up the majority of practical problems, while ignoring the complexities of arbitrary quantification. BOUNDER helps PLR [Sacks, 1987b] explore the qualitative behavior of dynamic systems, such as stability and periodicity. For example, suppose a linear system contains symbolic parameters. Given constraints on the parameters, one can use BOUNDER to reason about the locations of the system's poles and zeroes. PLR also enhances the performance of QMR [Sacks, 1985], a program that derives the *qualitative properties* of parameterized functions: signs of the first and second derivatives, discontinuities, singularities, and asymptotes.

BOUNDER consists of an inequality prover, a context manager, and four bounding algorithms: bounds propagation, substitution, derivative inspection, and iterative approximation. The prover uses the bounding algorithms to resolve inequalities, as described above. First, it reduces the original inequality to an equivalent but simpler one by canceling common terms and replacing monotonic functions with their arguments. For example, $x+1 \leq y+1$ simplifies to $x \leq y$, $-x \leq -y$ to $x \geq y$, and $e^x \leq e^y$ to $x \leq y$. The prover only cancels multiplicands whose signs it can determine by bounds propagation.

The context manager organizes constraint sets in the format required by the bounding algorithms. The bounding algorithms derive upper and lower bounds for a function over all points satisfying a constraint set. The context manager and bounding algorithms are described in the next two sections. The final two sections contain a review of literature and conclusions. I argue that current inequality provers are weak, brittle, or inefficient because they process all inputs uniformly, whereas BOUNDER avoids

[1]This research was supported (in part) by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine and National Institutes of Health Grant No. R24 RR01320 from the Division of Research Resources.

these shortcomings with its hierarchical strategy. While this paper discusses only inequality constraints and nonstrict inequalities, BOUNDER implements boolean combinations of inequality constraints and strict inequalities analogously.

II. The Context Manager

The context manager derives an upper (lower) bound for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ ($x \geq U$) with U free of x. It derives upper and lower bounds for x from an equality $L = R$ by reformulating it as $x = U$. Inequality manipulation may depend on the signs of the expressions involved. For example, the constraint $ax \leq b$ can imply $x \leq b/a$ or $x \geq b/a$ depending on the sign of a. In such cases, the context manager attempts to derive the relevant signs from other members of the constraint set using bounds propagation. If it fails, it ignores the constraint. Constraints whose variables cannot be isolated, such as $x \leq 2^x$, are ignored as well. The number of variables in a constraint is linear in its length and each variable requires linear time to isolate. Isolation may require deriving the signs of all the subexpressions in the constraint. Theorem 1 implies that this process takes linear time. All told, processing each constraint requires quadratic time in its length. Subsequent complexity results exclude this time.

Two pairs of functions form the interface between the context manager and the bounding algorithms. Given a variable x and a set of constraints C, the functions VAR-LB$_C(x)$ and VAR-UB$_C(x)$ return the maximum of x's numeric lower bounds in C and the minimum of its numeric upper bounds. The functions LOWER$_C(x)$ and UPPER$_C(x)$ return the maximum over all lower bounds, symbolic and numeric, and the minimum over all upper bounds. Both VAR-LB and LOWER derive lower bounds for x, whereas both VAR-UB and UPPER derive upper bounds. However, LOWER and UPPER produce tighter bounds then VAR-LB and VAR-UB because they take symbolic constraints into account. Examples of these functions appear in Table 1. All four functions run in constant time once the contexts are constructed.

Table 1: Bounds of $\{a \geq 1, b \leq 0, b \geq -2, ab \geq -4, c = b\}$

	VAR-LB	VAR-UB	LOWER	UPPER
a	1	∞	1	$-4/b$
b	-2	0	$\max\{-2, -4/a, c\}$	$\min\{0, c\}$
c	$-\infty$	∞	b	b

III. Bounding Algorithms

This section contains the details of the bounding algorithms. Each derives tighter bounds than its predecessor, but takes more time. Each invokes all of its predecessors for subtasks, except that derivative inspection never calls bounds propagation. The bounding algorithms define the extended elementary functions on the extended real numbers in the standard fashion, so that $1/\pm\infty = 0$, $\log 0 = -\infty$, $2^\infty = \infty$, and so on. Throughout this paper, "number" refers to an extended real number.

A. Bounds Propagation

The bounds propagation algorithm (BP) bounds a compound function by bounding its components recursively and combining the results. For example, the upper bound of a sum is the sum of the upper bounds of its addends. The recursion terminates when it reaches numbers and variables. Numbers are their own bounds, while VAR-LB and VAR-UB bound variables. Figures 1 and 2 contain the upper bound algorithm, UB$_C(e)$, for a function e over a set of constraints C. The lower bound algorithm, LB$_C(e)$, is analogous. One can represent e as an expression in its variables $x_1, \ldots, x_n$ or as a function $e(x)$ of the vector $x = (x_1, \ldots, x_n)$. From here on, these forms are used interchangeably. The TRIG-UB algorithm, not shown here, uses periodicity and monotonicity information to derive upper bounds for trigonometric functions and their inverses.

e is	UB$_C(e)$
a number	e
a variable	VAR-UB$_C(e)$
$a + b$	$ub_a + ub_b$
ab	$\max\{lb_a lb_b, lb_a ub_b, ub_a lb_b, ub_a ub_b\}$
a^b	EXPT-UB$_C(a, b)$
$\min\{a, b\}$	$\min\{ub_a, ub_b\}$
$\max\{a, b\}$	$\max\{ub_a, ub_b\}$
$\log a$	$\log ub_a$
$\lvert a\rvert$	$\max\{\lvert lb_a\rvert, \lvert ub_a\rvert\}$
trigonometric	TRIG-UB$_C(e)$

Figure 1: The UB$_C(e)$ algorithm; lb_e and ub_e abbreviate LB$_C(e)$ and UB$_C(e)$.

Case	EXPT-UB$_C(a, b)$
UB$_C(a) > 0$	$e^{\text{UB}_C(b \log a)}$
$b = \frac{p}{q}$ with p, q integers	
p, q odd and positive	$[\text{UB}_C(a)]^b$
p, q odd and $ub_a < 0$	$[\text{LB}_C(a)]^b$
p even	$e^{\text{UB}_C(b \log \lvert a\rvert)}$
else	∞
else	∞

Figure 2: The EXPT-UB$_C(a, b)$ algorithm

The correctness and complexity of BP are summarized in the theorem:

Theorem 1 *For any extended elementary function $e(x)$ and set of constraints C, bounds propagation derives numbers lb_e and ub_e satisfying*

$$\forall x. satisfies(x, S) \Rightarrow lb_e \leq e(x) \leq ub_e \qquad (1)$$

in time proportional to e's length.

The proof is by induction on e's length. It appears in a longer version of this paper [Sacks, 1987a], as do all subsequent proofs.

Bounds propagation achieves linear time-complexity by ignoring constraints among variables or multiple occurrences of a variable in an expression. It derives excessively loose bounds when these factors prevent all the constituents of an expression from varying independently over their ranges. For instance, the constraint $a \leq b$ implies that $a - b$ cannot be positive. Yet given only this constraint, BP derives an upper bound of ∞ for $a - b$ by adding the upper bounds of a and $-b$, both ∞. As another example, when no constraints exist, the joint occurrence of x in the constituents of $x^2 + x$ implies a global minimum of $-1/4$. Yet BP deduces a lower bound of $-\infty$ by adding the lower bounds of x^2 and x, 0 and $-\infty$. Subsequent bounding algorithms derive optimal bounds for these examples. Substitution analyzes constraints among variables and the final two algorithms handle multiple occurrences of variables. All three obtain better results than BP, but pay an exponential time-complexity price.

B. Substitution

The substitution algorithm constructs bounds for an expression by replacing some of its variables with their bounds in terms of the other variables. Substitution exploits all solvable constraints, whereas bounds propagation limits itself to constraints between variables and numbers. In our previous example, substitution derives an upper bound of 0 for $a - b$ from the constraint $a \leq b$ by bounding a from above with b, that is $a - b \leq b - b = 0$. Substitution is performed by the algorithms $\text{SUP}_C(e, H)$ and $\text{INF}_C(e, H)$, which calculate upper and lower bounds on e over the constraint set C in terms of the variable set H. When H is empty, the bounds reduce to numbers.

Figures 3 and 4 contain the SUP function and its auxiliary, SUPP. The auxiliary functions EXPT-SUP and TRIG-SUP are derived from BP's exponential and trigonometric bounding algorithms by replacing $\text{UB}_C(a)$ with $\text{SUP}_C(a, H)$, $\text{LB}_C(a)$ with $\text{INF}_C(a, H)$, and so on for b. The expression $v(e)$ denotes the variables contained in e and $f(b, a, H)$ abbreviates $v(b) - v(a) \subseteq H$. In the remainder of this section, we will focus on SUP. INF is analogous.

In step 1, SUP calculates the upper bounds of numbers and of variables included in H. It analyzes a variable, x, not in H by constructing an intermediate bound

$$B = \text{SUP}_C(\text{UPPER}_C(x), H \cup \{x\}) \qquad (2)$$

for x and calling SUPP to derive a final bound. If possible, SUPP derives an upper bound for x in H directly from the inequality $x \leq B$. Otherwise, it applies bounds propagation to B. For instance, the inequality $x \leq 1 - x$ yields a bound of $1/2$, but $x \leq x^2 - 1$ does not provide an upper bound, so SUPP returns $\text{UB}_C(x^2 - 1)$.

	e is	$\text{SUP}_C(e, H)$
1	$v(e) \subseteq H$	e
2	a variable	$\text{SUPP}_C(e, s(\text{UPPER}_C(e), H \cup \{e\}))$
3	$a + b$	
	3.1 $f(b, a, H)$	$s(a, H) + s(b, H)$
	3.2 else	$f(b, a, H)$
4	ab	
	4.1 $\text{LB}_C(a) \geq 0$	
	$f(b, a, H)$	$\max\{s(a, H)s(b, H), i(a, H)s(b, H)\}$
	else	$s(a\, s(b, H \cup v(a)), H)$
	4.2 $\text{UB}_C(a) \leq 0$	
	$f(b, a, H)$	$\max\{s(a, H)i(b, H), i(a, H)i(b, H)\}$
	else	$s(a\, i(b, H \cup v(a)), H)$
	4.3 else	$\max\{s(a, H)s(b, H), s(a, H)i(b, H),$ $i(a, H)s(b, H), i(a, H)i(b, H)\}$
5	a^b	$\text{EXPT-SUP}_C(a, b, H)$
6	$\min\{a, b\}$	$\min\{s(a, H), s(b, H)\}$
7	$\max\{a, b\}$	$\max\{s(a, H), s(b, H)\}$
8	$\log a$	$\log s(a, H)$
9	$\lvert a \rvert$	$\max\{\lvert i(a, H) \rvert, \lvert s(a, H) \rvert\}$
10	trig	$\text{TRIG-SUP}_C(e, H)$

Figure 3: The $\text{SUP}_C(e, H)$ algorithm. The symbols s and i abbreviate SUP_C and INF_C respectively.

	case	$\text{SUPP}_C(x, B)$
1	$x \notin v(B)$	B
2	$B = rx + A$	
	$r \in \Re$, $v \notin v(A)$	
	2.1 $r \geq 1$	∞
	2.2 $r < 1$	$\frac{A}{1-r}$
3	$B = \min\{C, D\}$	$\min\{\text{SUPP}_C(x, C), \text{SUPP}_C(x, D)\}$
4	$B = \max\{C, D\}$	$\max\{\text{SUPP}_C(x, C), \text{SUPP}_C(x, D)\}$
5	else	$\text{UB}_C(B)$

Figure 4: The $\text{SUPP}_C(x, B)$ algorithm

SUP exploits constraints among variables to improve its bounds on sums and products. If b contains variables that a lacks, but which have bounds in a's variables, SUP constructs an intermediate upper bound, U, for $a + b$ or ab by replacing b with these bounds. A recursive application of SUP to U produces a final upper bound. (Although not indicated explicitly in Figure 3, these steps are symmetric in a and b.) If a and b have the same variables, SUP bounds $a + b$ and ab by recursively bounding a and b and applying bounds propagation to the results. For example, given the constraints $c \geq 1$, $d \geq 1$, and $cd \leq 4$, SUP derives an intermediate bound of $3c/4$ for $c - 1/d$ by replacing $-1/d$ with $-c/4$, its upper bound in c. This bound is derived as follows:

$$\text{SUP}(-\frac{1}{d}, \{c\}) = -\text{INF}(\frac{1}{d}, \{c\}) = \frac{-1}{\text{SUP}(d, \{c\})} = -\frac{c}{4} \qquad (3)$$

SUP uses the recursive call $\text{SUP}(c, \{\}) = 4$ to derive a final bound of 3 for $c - 1/d$. The following theorem establishes the correctness of substitution:

Theorem 2 *For every extended elementary function e, variable set H, and constraint set C, the expressions* $i = \text{INF}_C(e, H)$ *and* $s = \text{SUP}_C(e, H)$ *satisfy the conditions:*

$$i \text{ and } s \text{ are expressions in } H \tag{4}$$

$$\forall x. satisfies(x, S) \Rightarrow i(x) \leq e(x) \leq s(x) \tag{5}$$

Substitution utilizes constraints among variables to improve on the bounds of BP, but ignores constraints among multiple occurrences of variables. It performs identically to BP on the example of $x^2 + x$, deriving a lower bound of $-\infty$. Yet that bound is overly pessimistic because no value of x minimizes both addends simultaneously. The last two bounding algorithms address this shortcoming.

C. Derivative Inspection

Derivative inspection calculates bounds for a function over a constraint set C from the signs of its partial derivatives. Let us define the *range* of x_i in C as the interval

$$X_i = [\text{INF}_C(x_i, \{\}), \text{SUP}_C(x_i, \{\})] \tag{6}$$

and the *range* of $x = (x_1, \ldots, x_n)$ in C as the Cartesian product $X = X_1 \times \cdots \times X_n$ of its components' ranges. Derivative inspection splits the range of a function $f(x)$ into subregions by dividing the range of each x_i into maximal intervals on which $\partial f/\partial x_i$ is non-negative, non-positive, or of unknown sign. The maximum upper bound over all subregions bounds f from above on X. This bound is valid over all points satisfying C by Theorem 2. Each region can be collapsed to the upper (lower) bound of x_i in every dimension i where $\partial f/\partial x_i$ is non-negative (non-positive) without altering f's upper bounds. An analogous procedure derives lower bounds.

Derivative inspection takes time proportional to the number of regions into which f's domain splits. For this reason, it only applies to functions whose partial derivatives all have finitely many zeroes in X. When the signs of all partial derivatives are known, derivative inspection yields optimal bounds directly, since all regions reduce to points. For example, it derives an optimal lower bound of $-1/4$ for $x^2 + x$ because the derivative of $x^2 + x$ is non-positive on $[-\infty, -1/2]$ and non-negative on $[-1/2, \infty]$. Otherwise, one must use a second bounding algorithm to calculate bounds on the non-trivial subregions. This two-step approach generally yields tighter bounds than applying the second algorithm directly on f's entire domain, since the subregions are smaller and often reduce to points along some dimensions.

D. Iterative Approximation

Iterative approximation, like derivative inspection, reduces the errors in bounds propagation and substitution caused by multiple occurrences of variables. Instead of bounding a function over its entire range directly, it subdivides the regions under consideration and combines the results. Intuitively, BP's choice of multiple worst case values for a variable causes less damage on smaller regions because all these values are less far apart. Figure 5 illustrates this idea for the function $x^2 - x$ on the interval $[0, 1]$. Part (a) demonstrates that BP derives an overly pessimistic lower bound on $[0, 1]$ because it minimizes both $-x$ and x^2 independently. Part (b) shows that this factor is less significant on smaller intervals: the maximum of the two lower bounds, $-3/4$, is a tighter bound for $x^2 - x$ on $[0, 1]$ than that of part (a). One can obtain arbitrarily tight bounds by constructing sufficiently fine partitions.

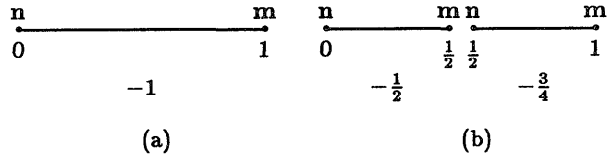

Figure 5: Illustration of iterative approximation on $[0, 1]$. The symbols m and n mark the values of x that minimize $-x$ and x^2. The numbers below are $\text{LB}(x^2 - x)$.

Iterative approximation generalizes interval subdivision to multivariate functions and increases its efficiency, using ideas from Moore [Moore, 1979] and Asaithambi et al. [Asaithambi *et al.*, 1982]. As an additional optimization, it bounds functions over the regions generated by derivative inspection, rather than over their entire domains. Let $f(x_1, \ldots, x_n)$ be continuously differentiable on a region X and let w_i denote the width of the interval X_i. For every positive ϵ, iterative approximation derives an upper bound for f on X that exceeds the least upper bound by at most ϵ within

$$\left(\frac{L}{\epsilon}\right)^n \prod_{i=1}^{n} w_i \tag{7}$$

iterations, where the constant L depends on f and X.

IV. Related Work

In this section, I discuss, in order of increasing generality, existing programs that derive bounds and prove inequalities. As one would expect, the broader the domain of functions and constraints, the slower the program. The first class of systems bounds linear functions subject to linear constraints. Valdés-Pérez [Valdés-Pérez, 1986] analyzes sets of *simple linear inequalities* of the form $x - y \geq n$ with x and y variables and n a number. He uses graph search to test their consistency in cv time for c constraints and v variables. Malik and Binford [Malik and Binford, 1983] and Bledsoe [Bledsoe, 1975] check sets of general linear constraints for consistency and calculate bounds on linear functions over consistent sets of constraints. Both methods require exponential time.[2] The former uses the

[2]The simplex algorithm often performs better in practice. Also, a polynomial alternative exists.

Simplex algorithm, whereas the latter introduces preliminary versions of BOUNDER's substitution algorithms. Bledsoe defines SUP, SUPP, INF, and INFF for linear functions and constraints and proves the linear version of Theorem 2. In fact, these algorithms produce *exact* bounds, as Shostak [Shostak, 1977] proves.

The next class of systems bounds nonlinear functions, but allows only range constraints. All resemble BOUNDER's bounds propagation and all stem from Moore's [Moore, 1979] *interval arithmetic*. Moore introduces the rules for bounding elementary functions on finite domains by combining the bounds of their constituents. His algorithm takes linear time in the length of its input. Bundy [Bundy, 1984] implements an *interval package* that resembles BP closely. It generalizes the combination rules of interval arithmetic to any function that has a finite number of extrema. If the user specifies the sign of a function's derivative over its domain, Bundy's program can perform interval arithmetic on it. Unlike BOUNDER's derivative inspection algorithm, it cannot derive this information for itself. Many other implementations of interval arithmetic exist, some in hardware.

Moore also proposes a simple form of iterative approximation, which Skelboe [Skelboe, 1974], Asaithambi et al. [Asaithambi *et al.*, 1982], and Ratschek and Rokne [Ratschek and Rokne, 1984, ch. 4] improve. BOUNDER's iterative approximation algorithm draws on all these sources.

Simmons [Simmons, 1986] handles functions and constraints containing numbers, variables, and the four arithmetic operators. He augments interval arithmetic with simple algebraic simplification and inequality information. For example, suppose x lies in the interval $[-1, 1]$. Simmons simplifies $x - x$ to 0, whereas interval arithmetic produces the range $[-2, 2]$. He also deduces that $x \leq z$ from the constraints $x \leq y$ and $y \leq z$ by finding a path from x to z in the graph of known inequalities. The algorithm is linear in the total number of constraints. Although more powerful than BOUNDER's bounds propagation, Simmons's program is weaker than substitution. For example, it cannot deduce that $x^2 \geq y^2$ from the constraints $x \geq y$ and $y \geq 0$.

Brooks [Brooks, 1981, sec. 3] extends Bundy's SUP and INF to nonlinear functions and argues informally that Theorem 2 hold for his algorithms. This argument must be faulty because his version of $\text{SUP}_H(e, \{\})$ recurses infinitely when e equals $x + 1/x$ or $x + x^2$, for instance. Brooks's program only exploits constraints among the variables of sums $rx + B$ and of products $x^n B$ with r real, x a variable of known sign, B an expression free of x, and n an integer. In other cases, it adds or multiplies the bounds of constituents, as in steps 3.1, 4.1.1, 4.2.1, and 4.3 of BOUNDER's SUP (Figure 3). These overly restrictive conditions rule out legitimate substitutions that steps 3.2, 4.1.2, and 4.2.2 permit. For example, BOUNDER can deduce that $1/x - 1/y \geq 0$ from the constraints $y \geq x$ and $x \geq 1$, but Brooks's algorithm cannot. On some functions and non-empty sets H, his algorithm makes recursive calls with H empty. This produces needlessly loose bounds and sometimes causes an infinite recursion.

Bundy and Welham [Bundy and Welham, 1979] derive upper bounds for a variable x from an inequality $L \leq R$ by reformulating it as $x \leq U$ with U free of x. If U contains a single variable, they try to find its global maximum, M, by inspecting the sign of its second derivative at the zeroes of its first derivative. When successful, they bound x from above with M. Lower bounds and strict inequalities are treated analogously. They use a modified version of the PRESS equation solver [Bundy and Welham, 1981] to isolate x. As discussed in section II, inequality manipulation depends on the signs of the expressions involved. When this information is required, they use Bundy's interval package to try to derive it. The complexity of this algorithm is unclear, since PRESS can apply its simplification rules repeatedly, possibly producing large intermediate expressions. BOUNDER contains both steps of Bundy and Welham's bounding algorithm: its context manager derives bounds on variables from constraints, while its derivative inspection algorithm generalizes theirs to multivariate functions. PRESS may be able to exploit some constraints that BOUNDER ignores because it contains a stronger equation solver than does BOUNDER.

The final class of systems consists of theorem provers for predicate calculus that treat inequalities specially. These systems focus on general theorem proving, rather than problem-solving. They handle more logical connectives than BOUNDER, including disjunction and existential quantification, but fewer functions, typically just addition. Bledsoe and Hines [Bledsoe and Hines, 1980] derive a restricted form of resolution that contains a theory of dense linear orders without endpoints. Bledsoe et al. [Bledsoe *et al.*, 1983] prove this form of resolution complete. Finally, Bledsoe et al. [Bledsoe *et al.*, 1979] extend a natural deduction system with rules for inequalities. Although none of these authors discuss complexity, all their algorithms must be at least exponential.

V. Conclusions

Current inequality reasoners are weak, brittle, or inefficient because they process all inputs uniformly. Interval arithmetic systems, such as Bundy's and Simmons's, run quickly, but generate exceedingly pessimistic bounds when dependencies exist among the components of functions. These dependencies are caused by constraints among variables or multiple occurrences of a variable, as discussed in Section III.A. The upper bound of $a - b$ given $a \leq b$ demonstrates the first type, while the lower bound of $x^2 + x$ given no constraints demonstrates the second. Each of the remaining systems is brittle because it takes only one type of dependency into account. Iterative approximation, suggested by Moore, and derivative inspection, performed in the univariate case by Bundy and Welham, address the second type of dependency, but ignore the first. Conversely,

substitution, used (in a limited form) by Brooks and Simmons, exploits constraints among variables, while ignoring multiple occurrences of variables. All these systems are inefficient because they apply a complex algorithm to every input without trying a simple one first.

BOUNDER overcomes the limitations of current inequality reasoners with its hierarchical strategy. It uses substitution to analyze dependencies among variables and derivative inspection and iterative approximation to analyze multiple occurrences of variables. Together, these techniques cover far more cases than any single-algorithm system. Yet unlike those systems, BOUNDER does not waste time applying overly powerful methods to simple problems. It tries bounds propagation, which has linear time-complexity, before resorting to its other methods. An inequality reasoner like BOUNDER should be an important component of future general-purpose symbolic algebra packages.

References

[Asaithambi *et al.*, 1982] N. S. Asaithambi, Shen Zuhe, and R. E. Moore. On computing the range of values. *Computing*, 28:225–237, 1982.

[Bledsoe, 1975] W. W. Bledsoe. A new method for proving certain Presburger formulas. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 15–21, 1975.

[Bledsoe and Hines, 1980] W. W. Bledsoe and Larry M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. In *Proceeding of the fifth conference on automated deduction*, Springer-Verlag, Les Arcs, France, July 1980.

[Bledsoe *et al.*, 1979] W. W. Bledsoe, Peter Bruell, and Robert Shostak. A prover for general inequalities. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 66–69, 1979.

[Bledsoe *et al.*, 1983] W. W. Bledsoe, K. Kunen, and R. Shostak. *Completeness results for inequality provers.* ATP 65, University of Texas, 1983.

[Brooks, 1981] Rodney A. Brooks. Symbolic reasoning among 3-d models and 2-d images. *Artificial Intelligence*, 17:285–348, 1981.

[Bundy, 1984] Alan Bundy. A generalized interval package and its use for semantic checking. *ACM Transactions on Mathematical Software*, 10(4):397–409, December 1984.

[Bundy and Welham, 1979] Alan Bundy and Bob Welham. *Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.* D.A.I. Working Paper 55, University of Edinburgh, Depatment of Artificial Intelligence, May 1979.

[Bundy and Welham, 1981] Alan Bundy and Bob Welham. Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–211, May 1981.

[Malik and Binford, 1983] J. Malik and T. Binford. Reasoning in time and space. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 343–345, August 1983.

[Moore, 1979] Ramon E. Moore. *Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics*, SIAM, Philadelphia, 1979.

[Ratschek and Rokne, 1984] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Halsted Press: a division of John Wiley and Sons, New York, 1984.

[Sacks, 1985] Elisha P. Sacks. Qualitative mathematical reasoning. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 137–139, 1985.

[Sacks, 1987a] Elisha P. Sacks. *Hierarchical inequality reasoning.* TM 312, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, 1987.

[Sacks, 1987b] Elisha P. Sacks. Piecewise linear reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, 1987.

[Shostak, 1977] Robert E. Shostak. On the SUP-INF method for proving Presburger formulas. *Journal of the ACM*, 24:529–543, 1977.

[Simmons, 1986] Reid Gordon Simmons. "Commonsense" arithmetic reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 118–124, American Association for Artificial Intelligence, August 1986.

[Skelboe, 1974] S. Skelboe. Computation of rational functions. *BIT*, 14:87–95, 1974.

[Valdés-Pérez, 1986] Raúl Valdés-Pérez. *Spatio-temporal reasoning and linear inequalities.* AIM 875, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1986.